RESEARCH STATEMENT

LIN, Yun (dcsliny@nus.edu.sg/llmhyy@gmail.com)

1 Research Overview

Software is ubiquitous in the modern society. Thus, software engineering, systematic methodologies to ensure reliable software development and maintenance, is of vital importance to modern civilization. Programs are complicated and hard to understand: program behaviors sometimes get unexpected, causing life and financial loss; the bugs can happen from time to time, taking hours and days for programmers to figure out the reason; their forms keep evolving, from sequential to concurrent, from desktop to web and mobile, and from hard-coded logics to data-driven AI model decisions, making the causality behind program execution even harder to interpret and comprehend.

My research revolves around designing explainable software engineering techniques. Specifically, I look for software bugs and locate their root causes with explanation, visualize why and how the decision of AI models are formed during training, and generate explanation/justification for the decision of cybersecurity system. My major contributions lie in:

- **R1. Search-based Testing (bug discovery):** Automating test case generation for revealing software bugs[1] (ESEC/FSE'21, ISSTA'20, ICSE'20, ICSE'18) [3, 6, 12, 16],
- **R2. Explainable Time-travelling Debugging (bug location/explanantion):** Locating software bugs with explanation on how and why it happens, which (1) recommends suspicious event (or program execution step)[2] and (2) explains regression fault localization based on program-trace alignment[3] (ICSE'17, ASE'18, TSE'19, FSE'19) [7, 8, 14, 15],
- **R3. Visualizing/Explaining How Deep Models are Trained:** Visualizing how deep classifiers are trained in the visible two dimensional space, specifically, why and how *classification landscape* and *sample representation* are formed during the training process[4] (AAAI'22) [17],
- **R4. AI as an Explanation (applied to security system):** Using AI techniques (i.e., computer vision) to justify the decision in cybersecurity product like phishing webpage detector[5] (USENIX Sec'21, USENIX Sec'22 (under review)) [1, 11].

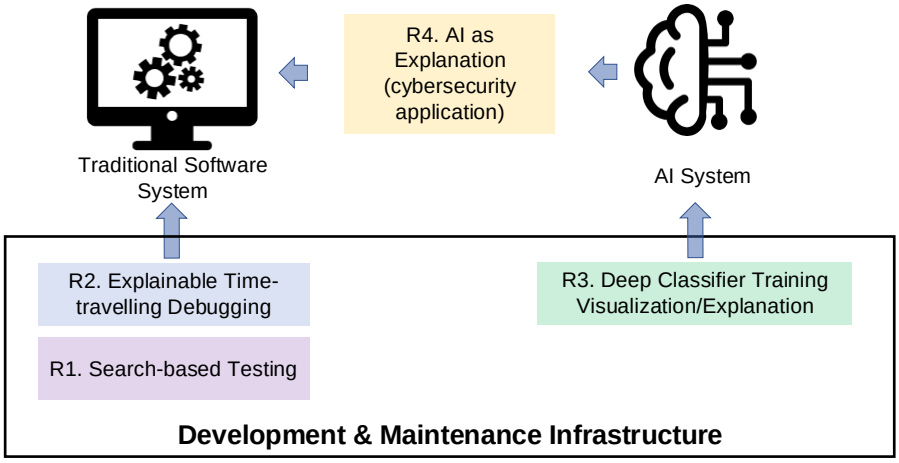


Figure 1: Research Landscape

Figure 1 shows the overall research landscape to build an infrastructure for developing, maintaining, and understanding both traditional and AI systems.

To support traditional software system, I develop tools and techniques for program testing and debugging, addressing the questions *how to locate the potential bugs* and *why the bug happens*. For discovering more potential

[1] https://youtu.be/Q9uQmMDvH4A
[2] video: `http://linyun.info/microbat/index.html`
[3] video: `https://youtu.be/r5F3dAq__Xo`
[4] https://sites.google.com/view/deepvisualinsight/home
[5] https://sites.google.com/view/phishpedia-site/home

bugs, I designed various techniques to improve state-of-the-art test generator (like EvoSuite) to cover more program branches. Moreover, our techniques are explanation-oriented. Our time-travelling debugging technique not only reports where is the root cause, it also additionally reports (1) under what the scenario a bug can happen, (2) how the program execution is different from the expected execution, and (3) how the defect is propagated along the execution to induce the final faulty observation.

To support understanding AI system, I develop visualization techniques to understand *how the model predictions are formed during training*. All data scientists know that there is an *invisible* high-dimensional classification boundary for any deep classifier. However, the boundaries are hard to track, understand, and debug given the dimensionality curse. Our work visualizes in two-dimensional space the evolving dynamics of classification landscape and sample embeddings.

Last, I also design AI solution to explain and justify the runtime behavior of a system. I proposed an AI-powered explanation system to justify why a webpage is believed to be malicious. Our system go beyond binary output (i.e., whether a webpage is malicious or not). Once a malicious webpage is reported, Phishpedia can justify the results with explanation of (1) which brand the attacker is trying to fake, (2) where on the webpage the credentials are to be stolen, and (3) what tricks the attacker is playing to deceive the victim.

In addition to the above interests, I also has research experience to automate the code recommendation (ASE'17, FSE'15) [2, 5], refactoring (FSE'16) [4], and program differencing (ICSE'14, ICSME'14) [9, 10] tasks. When developing tools to facilitate users (e.g., programmers), I have enriched experience to design/conduct user studies and simulated experiment (to mimic user behaviors) to evaluate the effectiveness of the tool designs.

2 Research Contributions

2.1 Search-based Testing (where is the bug?)

Background Search-based software testing (SBST) considers software testing as an optimization problem. Given an uncovered program branch and a test case t, SBST techniques define a measurement (e.g., branch distance) to evaluate how far the test case t is away from covering the branch. With the measurement as an objective function, SBST can keep evolving the test case t to minimize the objective function and cover the program branch eventually. Figure 2 shows an example. Existing test generator (e.g., EvoSuite and Randoop) and fuzzer (e.g., AFL) have proved their success to cover program branches and discovery software vulnerabilities in practice.

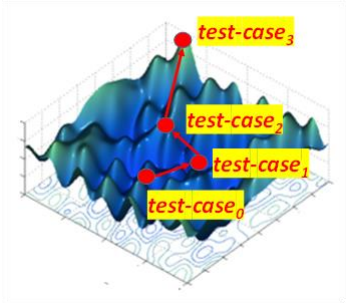


Figure 2: Search-based Software Testing

Problem The effectiveness of SBST is largely based on the assumption that the search space is *continuous* and *monotonous*. However, the assumption is not true in many more testing scenarios. As a result, many SBST approaches cannot achieve the optimal test cases and many program branches are still uncovered.

Contribution My work improves SBST in two folds, which technically improves the branch coverage of state-of-the-art tool EvoSuite by 5-7%. First, I propose a gradient recovery technique to reshape the noncontinuous and flat search space into a continuous and monotonous one (ISSTA'20) [6]. Specifically, when we find the search landscape is flat (e.g., flag problem in SBST), we use interprocedural program analysis technique to recover the search gradients. Second, I propose a test seed synthesis technique to generate a "shortcut" solution much closer to the global optimal solution (ESEC/FSE'21) [3]. Starting the search with a good initial seed can largely improve

the search efficiency of a lot of meta-heuristic search algorithms. Specifically, we transform the dataflow of a target program branch into a template of the object construction process. By this means, our approach can construct more legitimate object as inputs and achieve a much higher program branch coverage.

2.2 Explainable Time-travelling Debugging (why the bug happens?)

Background Debugging, or fault localization, is considered as one of the most time-consuming in software development. When a bug happens, programmers need to not only pinpoint the root cause, but also have a deep understanding so that they can fix it.

Problem Existing approaches like Spectrum-based Fault Localization (SBFL) assumes that we have a large number of failing and passing test cases, and use statistical approaches to estimate the bug potential of each line. Let alone how realistic that we have many failing and passing test cases for an individual bug, the reported the buggy lines still lack good explanation. Traditional record-and-replay debugging (or time-travelling debugging) can track the program execution. However, the trace length can be *huge*, the limited query support on the trace is still not convenient for programmers to investigate and explain the root cause. Typically, automated debugging approaches usually suffer from the fundamental **problem of specification missing, i.e., the specification of the code implementation is missing**.

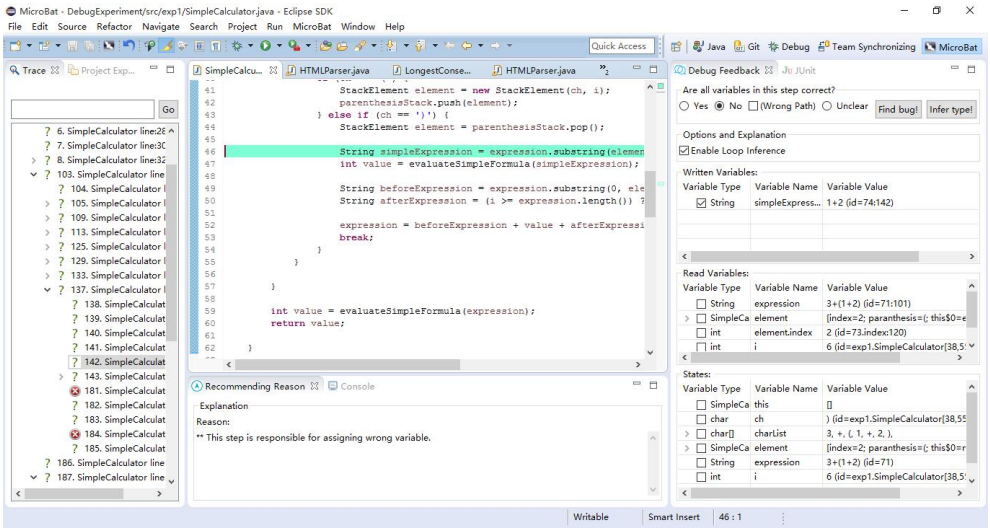


Figure 3: Feedback-based Debugging

Contribution My contribution for debugging research lies in three folds.
- Feedback on the trace as partial specification (ICSE'17): I propose a feedback-based debugging technique, Microbat, on the program execution. After the debugger record and replay the program execution trace, we regard the programmers' feedback on the trace as the partial specification. With a feedback on a step (e.g., read a wrong variable or visit a wrong step), the debugger can infer the casuality why it happens and recommend a step. The debugging process is designed as interactive and iterative: the programmers can further provide a feedback to let the debugger give another recommendation. In addition, our approach can also reason and estimate programmers' feedback to save the feedback efforts. See Figure 3 for the tool screenshot.
- Aligned trace as the explanation (TSE'19): I propose a trace-alignment based technique to debug regression bugs [13]. A regression is a bug which makes a working function fail. It can be presented in the form of $\langle P, P', t\rangle$ where the test case t passes in the old program P' but fails in the current program P. Taking the execution of P' as the reference, we can infer whether any execution steps of P are correct or not. We propose a trace-alignment technique to match the executions of P and P' in linear time complexity. Based on the trace difference, we can synthesize feedbacks on both trace such as *a step reads a wrong variable* or *a step should not happen*. With those feedback, we can use Microbat to track from the fault to the root cause. The process of bug tracking on both traces forms the regression explanation. Figure 4 shows the tool screenshot. The tool allows the user to interactively and iteratively explore the causality of a regression fault. The source code and demo is available at https://github.com/llmhyy/tregression.

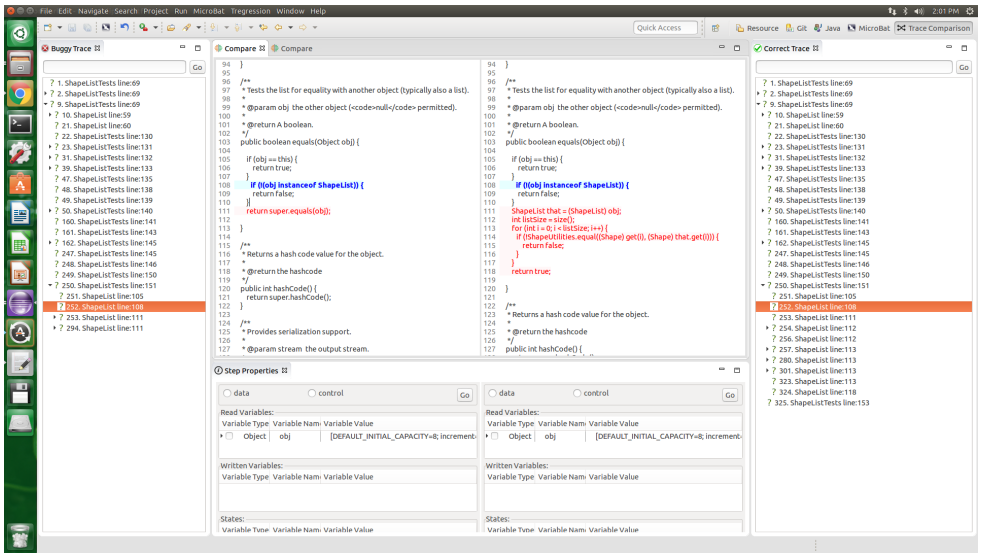


Figure 4: Trace-alignment Based Debugging

- AI-powered recommendation for buggy trace step (ASE'18): By building a dataset of aligned traces, I further propose a AI-powered technique to predict the fault steps on the trace. Specifically, I design a step embedding technique and feed step representation into a neural network. The approach achieves good performance over 3000 mutated bugs assimilating the bugs on Defects4j repository.

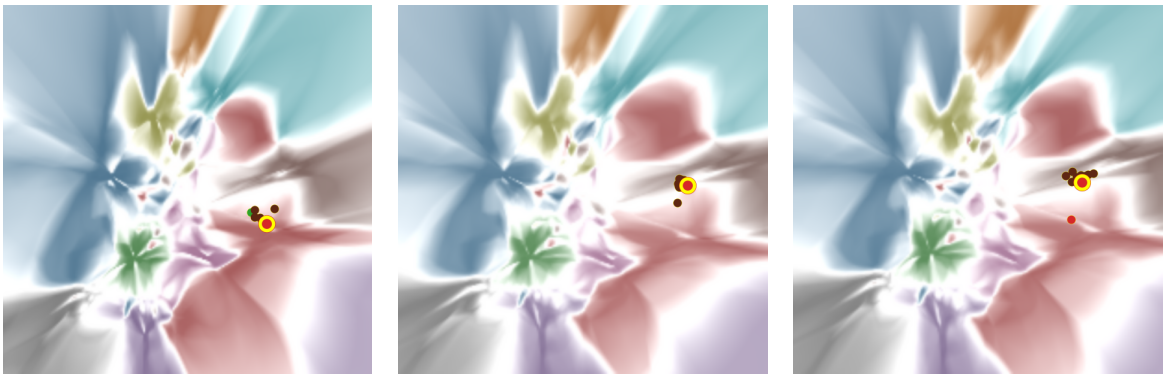
2.3 Causal Visualization for AI Models (why and how the model prediction forms?)

Background and Problem Deep learning models are widely used for classification tasks. Once the model's performance cannot meet the expectation, it takes data scientists and programmers great efforts to understand the root cause. Existing explainable AI techniques are proposed to answer why a sample will be predicted as a certain class, but very few techniques can answer *why the prediction are formed during the training?*.

Contribution I propose a technique, DeepVisualInsight, to visualize how the training samples and their formed classification boundary/landscape are evolved during the training process (AAAI'22) [17]. DeepVisualInsight is a time-travelling (or record-and-replay) visualization for deep classifiers. Moreover, we propose the spatial and temporal properties which needs to be satisfied by all the future time-travelling visualization. Technically, we design approaches to (inverse-)project between high-dimensional and low-dimensional space, while preserving a set of required mathematical properties.

Figure 5 shows our visualization of an adversarial training process on CIFAR-10 dataset. Each point represents a sample and each color represents a class. The colors of points represent the labels of samples, and the color of a region represent a predicted class. For example, a point in red (class cat) located in brown (class dog) territory indicates that it is labelled as cat but classified as dog. Moreover, the color shade indicates the confidence of prediction, unconfident regions (i.e. classification boundaries) are visualized as white regions. Overall, the classification region and boundaries form the *classification landscape*. Here, *the model fitting process is visualized by the process of (1) classification boundary being reshaped and (2) those data points being pulled towards the territory of the corresponding colors*.

Figure 5 shows that DeepVisualInsight manifests (1) the boundary reshaping process when the model is adapting new adversarial and training samples, and (2) the process of trade-off being made between adversarial robustness and testing accuracy. For clarity, we show one testing point (large red point with yellow edge) and its ten nearest neighbour adversarial points (in brown) in Figure 5. During adversarial training, (1) the adversarial points are gradually pulled to their color-aligned territory, while (2) the testing point is also gradually "pulled" away from its color-aligned territory to the territory of its adversarial neighbours. Such trade-off is formed gradually. DeepVisualInsight tool can further visualize the process as animation. In addition, it supports samples and iteration queries for users to observe the dynamics of interested samples and iterations, gaining deep insights into the model training process.



(a) Iteration 1
- adv acc 51.3%
- testing acc 92.3%

(b) Iteration 2
- adv acc 67.8%
- testing acc 90.3%

(c) Iteration 3
- adv acc 68.8%
- testing acc 89.9%

Figure 5: Adversarial training process: dynamics of one testing point and its ten neighbouring adversarial points (adv acc stands for adversarial accuracy, test acc stands for testing accuracy)

2.4 AI as an Explanation (applied in security system, why the security system make such decision?)

Background and Problem Phishing is a prevalent cyber-attacks causing loss of critical information. However, existing phishing defense system can only report binary results, i.e., whether phishing or not.

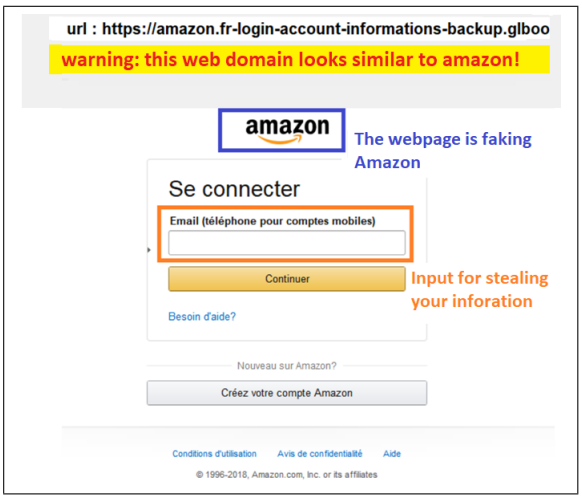


Figure 6: AI-generated Explanation for Phishing Detection

Contribution I propose an AI-powered explainable approach for phishing detection system, which generates explanation to justify its decision (USENIX Sec'21) [1]. Technically, we transform phishing detection problem as a computer vision problem of object detection and pattern recognition. The object detection technique can locate the logo information on the screenshot, while the pattern recognition technique (here, we use a Siamese neural network) can recognize which logo is used on the screenshot. By this means, we use AI techniques to extract the brand intention from the screenshot of a webpage. By comparing the identified brand and the domain of a webpage, we can decide why an URL/webpage is phishing, along with its explanation. Figure 6 shows our generated explanation for the phishing detection application. The explanation allows us to detect phishing webpages even more precisely. We deployed Phishpedia with a crawler and can discover about 50 new zero-day phishing websites every day.

2.5 Others

In addition to the above work, I also have enriched experience in:

- **Code Recommendation:** Approaches to recommend code edits on pasted code [5] and reusable implicit design to automate the generation of a similar new feature [2].
- **Program Differencing Algorithm:** Approaches to report code clone differences [10] and their summaries [9].
- **Architectural Refactoring:** Approaches to suggest and recommend architectural refactorings based on user-provided reflexion model [4].

The above approaches are usually designed with user interaction and incorporate user feedback to improve the tool performance.

3 Research Philosophy

My research philosophy involves tackling important problem, delivering practical tools, and building strong collaboration.

Tackling Important and Challenging Problems. Software engineering is usually considered to serve the benefits of software engineers. In my research, I would like to extend the application of software engineering techniques, such as dynamic and static program analysis, to areas such as AI, security, and HCI so that my research can gain more impact and serve more people in the world.

My research will investigate and target for

- What techniques can *fundamentally* change or improve the efficiency of code development and validation?
- How existing software engineering techniques, e.g., program analysis, software testing, etc., can help improve the developing and maintaining AI programs?
- How program analysis techniques can help discover the vulnerabilities of critical software systems?

Any of the envisioned breakthroughs for above questions can bring great benefits to both academic and industrial communities. Touching challenging problem is always my pursuit during my research work. For now, I expect my work of feedback-based time-travelling debugging framework have the potential to fundamentally change how programmers debug their programs. How to design a good debugger to intuitively visualize how the program works can benefits both professional, armature, and beginner programmers. Moreover, good debugging tools for AI programs can benefit a chain of users, from programmers to end users, in the world. Thus, I also heavily investigate how software engineering and program analysis techniques can help developing AI programs and how AI techniques can address fundamental software engineering research problems.

Delivering Practical Tools. Being practical is my another research philosophy. I aim to make my research useful in practice and lead to potential economic and market value. During my PhD and Postdoc training, each of my previous research work is supported by my developed open source Eclipse-plugin project. Those tools range from code clone differencing tool, code refactoring tool, to software debugger and program testing tools. More details can be check in http://linyun.info/tools.html.

For now, I am keeping building a flexible trace collection agent, which supporting time-travelling features of debugging. Once we can solve the scalability issue, the technique has the potential to change the debugging style of programmers. After being a faculty, I would like to deliver more solid research prototypes for the benefits of both research community and future industrial collaborators.

Building Strong Collaboration. Good research idea can be sparked by discussing with different people from different fields. Moreover, good research gains its impact by being applied in industry and many experimental ideas should withstand the test of industrial application. Therefore, I will build more collaboration with people from both academia and industry.

In the past few years, I established collaboration with Prof. Dong Jin Song (full professor at National University of Singapore, my supervisor), Prof. Gordon Fraser (full professor at University of Passau), Prof. Ivan Beschastnikh (associate professor at University of British Columnbia), Prof. David Rosenblum (chair professor at George Mason University), Prof. Sun, Jun (who is an Associate Professor in Singapore Management University), Prof. Liu, Yang

(full professor in Nanyang Technological University), Prof. Xing, Zhenchang (senior lecturer in Australian National University), Prof. Cai, Yuanfang (who is an Associate Professor in Drexcel University, USA),

Now, I am strengthening my connection with these excellent researchers and building more connection during my attending at different top-tier software engineering conference. Moreover, during my Ph.D. study and postdoctoral training, I also help collaborate with research lab in Huawei (in Shenzhen and Singapore). After getting the faculty position, I can conduct more research cooperations and help the growth of the research community and industry.

4 Research Visions

4.1 Trace-Travelling Based Root Cause Analysis System

One long-running research vision is to enhance and construct more comprehensive explainable time-travelling root-cause analysis framework, with which users can locate and recommend root cause interactively and efficiently. The ambition is to change the programmers' debugging convention and improve their efficiency in a significant way. Based on existing work, I plan to construct a more systematic framework, as illustrated in Figure 7. It has the following functionalities:

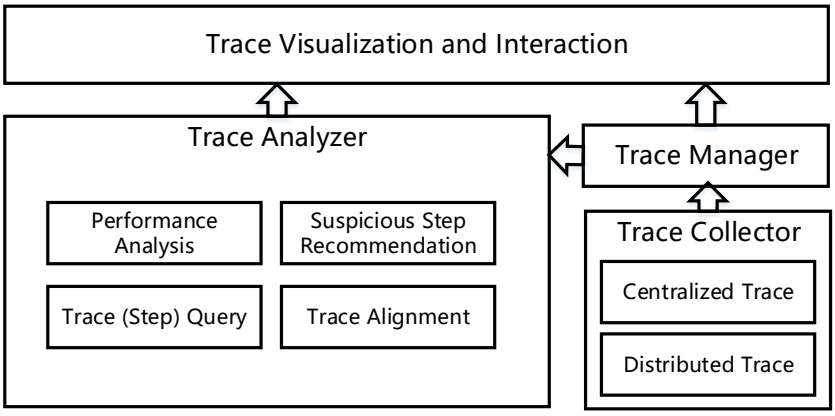


Figure 7: Trace-Travelling Based Root Cause Analysis System

- **More Powerful and Configurable Trace Collector.** In current stage, I built the trace model from execution trace by monitoring the program execution through code instrumentation. The new trace collector need to monitor a larger variety of program events such as thread creation, thread switch, etc. Based on the needs, the collector can be configured to keep the most interested events. The trace model can capture program features of concurrent and distributed programs. Moreover, once the trace model becomes more complicated, we will need new solutions to collect and model the trace with regard to its scalability and high performance.

  Problems to explore:

  - While recording all the execution can be expensive, how to adaptively infer the most important steps (or events) for different debugging applications?
  - How can we replicate the execution for various non-deterministic programs?

- **Trace Manager.** When the program runs for a long time, the generated trace model would be large accordingly. In this regard, I will develop a trace storage management system to record the trace and support expressive query language for users to search steps among the trace. The Trace Manager should be able to efficiently (1) capture the generated trace during program execution, (2) store the trace in disks, and (3) support programmers' query efficiently. One vision is to collect a great number of traces of both buggy and its fixed version and store them in the trace database. Thus, we can design machine learning algorithm for discovering potential runtime trace pattern for facilitating various software engineering tasks.

  Problems to explore:

  - How to design a solution to efficiently store and query program execution?

  - Whether we need to design a new query language to facilitate the trace query?

- **Trace Analyzer.** Based on the stored program executions, we can derive a series of trace-based analysis including performance analysis (e.g., whether some steps are unnecessarily executed), suspicious step recommendation (e.g., a more advanced AI-powered to infer some steps may read wrong variable or visit the wrong code), trace query (e.g., an efficient query system to retrieve user interested steps from the execution database), and trace alignment (e.g., derive step correctness by trace alignment technique on sequential, concurrent, and distributed programs).

  Problems to explore:

  - How can we infer mistaken user feedback?
  - How can we recommend faulty execution steps in a more efficient and accurate way?
  - How can we leverage program execution information to optimize the runtime efficiency?

- **Trace Visualization and Interaction.** In the long run, more variety of feedback types will be supported, which requires new reasoning technique for speeding up the debugging process. In addition, new trace visualization technique will be considered for providing a better UI for programmers. The programmers can provide more varieties of feedback on the new visualized UI and recommendations can be manifested more intuitively for programmers to locate the bugs.

  Problems to explore:

  - What is the most intuitive solution to take users' feedback?
  - How to design the user interface to explore the (concurrent/distributed) traces, especially when traces are long and many?

4.2 Visualizing Model Training Causality

The world is investing on AI projects dramatically, which will incur a soar on the number of AI developers. Therefore, how to facilitate debugging and testing on AI programs will be an important challenge. My research interest on traditional debugging can be shifted to AI program debugging.

Different from traditional program, AI program such as deep learning techniques usually aggregates inputs from a large corpus of training data and its behaviours are usually nondeterministic. In this light, the traditional program analysis techniques such as data or control flow analysis may not work anymore. Traditional debugging cares about how to diagnose the causality chain from the revealed fault to its root cause. The chain is usually fixed and deterministic. However, the "chains" in AI programs are usually overwhelming.

Based on my DeepVisualInsight work, my future research will further the investigation on how to build and visualize abstract causality chain when training a machine learning model so as to help developers localize the fault from a big picture instead of struggling with low-level details. Specifically, my follow-up works aim to answer the following questions:

- **Causality Construction:** How to attribute a training event (e.g., a sample is predicted with low confidence)? To dataset, model architecture, or hyberparameters?
- **Scalability:** How can we visualize and parse the training samples in a large scale?
- **Model Variety:** How can we visualize training process for more comprehensive AI model architecture such as transformer, LSTM, autoencoder, etc.?

4.3 Explainable Tools and Systems by Design.

While people are investigating generating explanation for AI, very few researchers notice how to use AI to generation explanation for software systems or construct an explainable system by design. We have some attempt to generate explanation based on AI solutions (the Phishpedia work [1]).

I argue that, explainable AI system shall NOT only focus on feedback causality, i.e., inventing explainable technique for some black-box technique. Instead, we shall focus on feedforward causality, i.e., constructing explainable software systems by design. In the future research, I will focus on AI-powered explainable tools and methodologies in the application of software engineering, security, and AI tasks. The solutions shall be explainable by design, which provides more friendly interaction with the system, and more convincing results for user to make decision.

5 Summary

In summary, I am looking forward to pursuing an academic career in your prestigious school. During my Ph.D. and Postdoc phase, each of the work takes fundamental effort to deeply go to research problem, build sophisticated tool, and the publication always has a high chance to be accepted in top venue. With the supported resources in your university, I believe I can build a strong research group with much higher productivity, grow up to be an influential scholar, and win good reputation for the university in the next 3-5 years.

References

[1] Yun Lin, Ruofan Liu, Dinil Mon Divakaran, Jun Yang Ng, Qing Zhou Chan, Yiwen Lu, Yuxuan Si, Fan Zhang, and Jin Song Dong. Phishpedia: A hybrid deep learning based approach to visually identify phishing webpages. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[2] Yun Lin, Guozhu Meng, Yinxing Xue, Zhenchang Xing, Jun Sun, Xin Peng, Yang Liu, Wenyun Zhao, and Jinsong Dong. Mining implicit design templates for actionable code reuse. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017, pages 394–404, Piscataway, NJ, USA, 2017. IEEE Press.

[3] Yun Lin, You Sheng Ong, Jun Sun, Gordon Fraser, and title=Graph-based Seed Object Synthesis for Search-Based Unit Testing year=2021 Jin Song Dong, booktitle=ESEC/FSE.

[4] Yun Lin, Xin Peng, Yuanfang Cai, Danny Dig, Diwen Zheng, and Wenyun Zhao. Interactive and guided architectural refactoring with search-based recommendation. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 535–546, New York, NY, USA, 2016. ACM.

[5] Yun Lin, Xin Peng, Zhenchang Xing, Diwen Zheng, and Wenyun Zhao. Clone-based and interactive recommendation for modifying pasted code. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 520–531, New York, NY, USA, 2015. ACM.

[6] Yun Lin, Jun Sun, Gordon Fraser, Ziheng Xiu, Ting Liu, and Jin Song Dong. Recovering fitness gradients for interprocedural boolean flags in search-based testing. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2020, page 440–451, New York, NY, USA, 2020. Association for Computing Machinery.

[7] Yun Lin, Jun Sun, Lyly Tran, Guangdong Bai, Haijun Wang, and Jinsong Dong. Break the dead end of dynamic slicing: Localizing data and control omission bug. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, pages 509–519, New York, NY, USA, 2018. ACM.

[8] Yun Lin, Jun Sun, Yinxing Xue, Yang Liu, and Jinsong Dong. Feedback-based debugging. In *Proceedings of the 39th International Conference on Software Engineering*, ICSE '17, pages 393–403, Piscataway, NJ, USA, 2017. IEEE Press.

[9] Yun Lin, Zhenchang Xing, Xin Peng, Yang Liu, Jun Sun, Wenyun Zhao, and Jinsong Dong. Clonepedia: Summarizing code clones by common syntactic context for software maintenance. In *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, volume 00, pages 341–350, Sept. 2014.

[10] Yun Lin, Zhenchang Xing, Yinxing Xue, Yang Liu, Xin Peng, Jun Sun, and Wenyun Zhao. Detecting differences across multiple instances of code clones. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 164–174, New York, NY, USA, 2014. ACM.

[11] Ruofan Liu, Yun Lin*, Xianglin Yang, Siang Hwee Ng, Dinil Divakaran, and Jin Song Dong. Inferring phishing intention via webpage appearance and dynamics: A deep vision based approach. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2022 (under review).

[12] Tai D Nguyen, Long H Pham, Jun Sun, Yun Lin, and Quang Tran Minh. sfuzz: An efficient adaptive fuzzer for solidity smart contracts. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 778–788, 2020.

[13] Haijun Wang, Yun Lin, Yang Liu, Zijiang Yang, Jun Sun, Jin Song Dong, Qinghua Zheng, and Ting Liu. Explaining regressions via alignment slicing and mending. In *IEEE Transaction on Software Engineering (TSE)*, 2019.

[14] Haijun Wang, Yun Lin, Zijiang Yang, Jun Sun, Yang Liu, Jin Song Dong, Qinghua Zheng, and Ting Liu. Explaining regressions via alignment slicing and mending. *IEEE Transactions on Software Engineering*, pages 1–1, 2019.

[15] Haijun Wang, Xiaofei Xie, Shang-Wei Lin, Yun Lin, Yuekang Li, Shengchao Qin, Yang Liu, and Ting Liu. Locating vulnerabilities in binaries via memory layout recovering. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 718–728, 2019.

[16] Xinyu Wang, Jun Sun, Zhenbang Chen, Peixin Zhang, Jingyi Wang, and Yun Lin. Towards optimal concolic testing. In *Proceedings of the 40th International Conference on Software Engineering*, pages 291–302, 2018.

[17] Xianglin Yang, Yun Lin*, Ruofan Liu, Zhenfeng He, Chao Wang, Jin Song Dong, and Hong Mei. Deepvisualinsight: Time-travelling visualization for spatio-temporal causality of deep classification training. In *2022 36th AAAI Conference on Artificial Intelligence (AAAI)*, 2022.